Electronic Engineering Programming

Lecturer: Professor Kim Jin Young

Dang Thanh Vu – 197796

1. Review

1.1. Expected discounted return

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{T} \gamma^k R_{t+k+1}$$

$0 \leq \gamma \leq 1$: discount rate. $T \to 0$ continuing task, $T = a < \infty$ episodic task.

1.2. Markov Property

$$P(s', r|s, a) = Pr\{R_{t+1} = r, S_{t+1} = s'|S_t = s, A_t = a\}$$

Where $s', s \in S$ (state space), $r \in R \subset \mathbb{R}, a \in A(s)$ (actions in state s)

1.3. Markov Decision Processes

- Expected rewards for state-action pairs:

$$r(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r \in R} \sum_{s' \in S} p(s', r|s, a)$$

- State-transition probabilities:

$$p(s'|s, a) = Pr\{S_{t+1} = s'|A_t = a, S_t = s\} = \sum_{r \in R} p(s', r|s, a)$$

- Expected rewards for the state-action-next-state:

$$r(s, a, s') = \mathbb{E}[R_{t+1}|S_t, A_t = a, S_{t+1} = s'] = \frac{\sum_{r \in R} r p(s', r|s, a)}{p(s'|s, a)}$$

- The value function is the expected return when starting in s and following π thereafter:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right]$$

- Action value function:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right]$$

Where $\pi(a|s)$ is a policy, the probability of taking action a when in state s. $a = \pi(s)$ is a mapping from state space to action space.

1.4. Bellman Equation

- Bellman equation for π

$$v_\pi(s) = \sum_{a, s', r} p(s', r|s, a)\pi(a|s)[r + \gamma v_\pi(s')] = \sum_{r, s'} [r + \gamma v_\pi(s')]p(r, s'|s)$$

Proof

Law of total expectation: $\mathbb{E}\big[\mathbb{E}[X|Y]\big] = \mathbb{E}[X]$, where X, Y are random variables.

$$\mathbb{E}\big[\mathbb{E}[X|Y]\big] = \int \mathbb{E}[X|Y]p(y)dy = \int\int xp(x|y)p(y)dxdy = \int x\left(\int p(x|y)p(y)dy\right)dx$$
$$= \int x\left(\int p(x,y)dy\right)dx = \int xp(x)dx = \mathbb{E}(X)$$

The discrete case is analogous when substituting integral by summation.

Corollary: $\mathbb{E}_{Z|Y}\big[\mathbb{E}[X|Y,Z]\big] = E[X|Y]$

We have,

$$\mathbb{E}_{S_{t+1}|S_t}\left[\mathbb{E}_{G_{t+1}}[G_{t+1}|S_t = s, S_{t+1} = s']\right] = \sum_{s'}\sum_{g} gp(g|s',s)p(s'|s)$$
$$= \sum_{s'}\sum_{g} gp(g|s')p(s'|s) \ \text{(Markov property)}$$

$$= \sum_{s'} \mathbb{E}[G_{t+1}|S_{t+1} = s']p(s'|s) = \sum_{s'} v(s')p(s'|s) = \mathbb{E}_{S_{t+1}}[v(S_{t+1})|S_t = s]$$

Therefore,

$$\mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\big[\textstyle\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\,\big|S_t = s\big] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s]$$
$$= \mathbb{E}_\pi[R_{t+1}|S_t = s] + \gamma\mathbb{E}_\pi[G_{t+1}|S_t = s]$$
$$= \mathbb{E}_\pi[R_{t+1}|S_t = s] + \gamma\mathbb{E}_\pi\big[\mathbb{E}_\pi[G_{t+1}|S_t = s, S_{t+1} = s']\big]$$
$$= \mathbb{E}_\pi[R_{t+1}|S_t = s] + \gamma\mathbb{E}_\pi[v_\pi(S_{t+1})|S_t = s]$$

Because

$$\mathbb{E}_\pi[R_{t+1}|S_t = s] = \sum_r rp(r|s) = \sum_r r\sum_{s'} p(r,s'|s)$$

$$\mathbb{E}[v_\pi(S_{t+1})|S_t = s] = \sum_{s'} v_\pi(s')p(s'|s) = \sum_{s'} v_\pi(s')\sum_r p(r,s'|s)$$

We have the Bellman Equation

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \sum_r r\sum_{s'} p(r,s'|s) + \gamma\sum_{s'} v_\pi(s')\sum_r p(r,s'|s) = \sum_{r,s'}\big(r + \gamma v_\pi(s')\big)p(r,s'|s)$$
$$= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$$

With the same procedure, we have

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a]$$
$$= \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

1.5. Optimal Value

- Optimal state-value function

$$v_*(s) = \max_\pi v_\pi(s)\,, \forall s \in S$$

- Optimal action-value function

$$q_*(s,a) = \max_\pi q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]\,, \forall s \in S, a \in A(s)$$

- Bellman optimality equation

$$v_*(s) = \max_{a \in A(s)} q_{\pi_*}(s,a) = \max_{a \in A(s)} \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

- Policy evaluation

$$v_{k+1} = \sum_{a,s',r} \pi(a|s)p(s',r|s,a)[r + \gamma v_k(s')]$$

- Policy improvement theorem

$$q_\pi\big(s,\pi'(s)\big) \geq v_\pi(s) \Rightarrow v_{\pi'}(s) \geq v_\pi(s), \forall\, s \in S$$

Where π, π' is any pair of deterministic policies.

Proof

Assume that

$$q_\pi\big(s,\pi'(s)\big) = \sum_{s',r} p(s',r|s,\pi'(s))[r + \gamma v_\pi(s')] = \sum_{s',r} p_{\pi'}(s',r|s)[r + \gamma v_\pi(s')]$$
$$= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$$

$$v_\pi(s) \leq q_\pi\big(s,\pi'(s)\big) = \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \leq \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma q_\pi\big(S_{t+1},\pi'(S_{t+1})\big)\big|S_t = s\big]$$
$$= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2})|S_{t+1}]|S_t = s]$$
$$= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2})|S_t = s] \text{ (Markov property)}$$

Using $\mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2})|S_t = s]$, by induction we can conclude

$$v_\pi(s) \leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots |S_t = s] = v_{\pi'}(s)$$

- Greedy policy

$$\pi'(s) = \text{argmax}_a\, q_\pi(s,a) = \text{argmax}_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

$$v_{\pi'}(s) = \max_a q_\pi(s,a) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

2. Jack's Car Rental

2.1. Problem Modeling

- Time steps: t (days)
- States: tuples of the numbers of cars in each location, $S = (S_1, S_2) \in \{0,1,\dots,20\}^2$
- Actions: move cars from one location to another location, $a \in \{-5,-4,-3,-2,-1,0,1,2,3,4,5\}$.

$$\text{sign}(a) = \begin{cases} 0 \text{ move cars from the first to the second location} \\ 1 \text{ move cars from the second to the first location} \end{cases}$$

$$\text{abs}(a) = \text{the number of cars which would be moved}$$

- Reward: $R = 10 \times X_{\text{rented}} - 2 \times X_{\text{moved}}$. Each rented car in each location will worth 10, each moving car will cost 2.

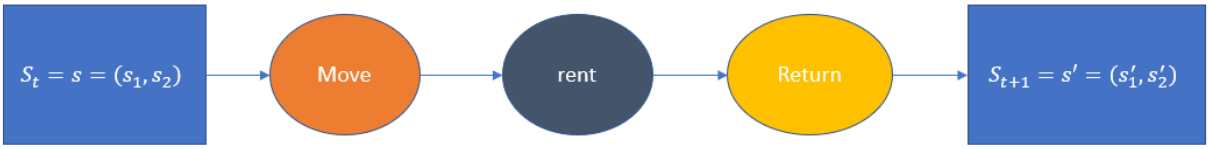

Figure 1. State Transition

THE NUMBER OF CARS IN THE FIRST LOCATION

THE NUMBER OF CARS IN THE SECOND LOCATION	0	1	…	20
0				
1				
…				
20				

Figure 2. State matrix. The state-value and taken action of each state will be stored under a 2d-array.

2.2. Algorithm

To use the "Value iteration" algorithm, we need to calculate

$$V(s) = \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$$

Which depends on the probability $p(s', r|s, a)$. Precisely,

$$\Pr\{R, S'_1, S'_2 | S_1, S_2, X^{move}\}$$

Where

$$R = 10 * \left(X_1^{rented} + X_2^{rented}\right) - 2 * abs(X^{move})$$

$$S'_1 = S_1 - X_1^{rented} + X_1^{return} + X^{move}$$

$$S'_2 = S_2 - X_2^{rented} + X_2^{return} - X^{move}$$

The probability $\Pr\{R, S'_1, S'_2 | S_1, S_2, X^{move}\}$ can be rewritten as

$$\Pr\{X_1^{rented}, X_2^{rented}, X_1^{return}, X_2^{return} | S_1, S_2, X^{move}\}$$

Because R, S'_1, S'_2 only depend on the random variables $X_1^{rented}, X_2^{rented}, X_1^{return}, X_2^{return}$ and the parameters S_1, S_2, X_{move}. Besides, rented and returned cars at each location are mutually exclusive independent given S_1, S_2, X^{move}. Therefore, we have

$$\Pr\left\{X_1^{rented}, X_2^{rented}, X_1^{return}, X_2^{return} \middle| S_1, S_2, X^{move}\right\} = \Pr\left\{X_1^{return}, X_1^{rented} \middle| S_1, X^{move}\right\} \Pr\left\{X_2^{return}, X_2^{rented} \middle| S_2, X^{move}\right\}$$

Each probability term is calculated by the following rule, Figure 3 and 4

$$\Pr\left\{X_1^{return}, X_1^{rented} \middle| S_1, X^{move}\right\} = \begin{cases} Poisson\left(x \middle| \lambda_1^{rented}\right) Poisson\left(y \middle| \lambda_1^{returned}\right) \\ \text{if } x \le s_1 + x^{move} \text{ and } 0 \le y \le 20 - s_1 + x \\ 0 \qquad \text{otherwise} \end{cases}$$

$$\Pr\left\{X_2^{return}, X_2^{rented} \middle| S_2, X^{move}\right\} = \begin{cases} Poisson\left(x \middle| \lambda_2^{rented}\right) Poisson\left(y \middle| \lambda_2^{returned}\right) \\ \text{if } x \le s_2 - x^{move} \text{ and } 0 \le y \le 20 - s_2 + x \\ 0 \qquad \text{otherwise} \end{cases}$$

```c
float Poisson(int n, int lambda)
{
    if(n >= 0)
    {
        float fac = 1.0;
        int i = 1;
        while(i <= n) {fac = fac * i; i++;}
        float p = pow(lambda, n)/fac*exp(-lambda);
        return p;
    }
    return 0;
}
```

Figure 3. C program to calculate the Poisson probability

```
for(int rented = 0; rented <= start; rented++)
{
    int returned = end - start + rented;

    if(returned >= 0)//feasible : prob != 0
    {
        //rental probability
        if(rented != start)
        {
            rental_prob = Poisson(rented, rent_lam);
        }
        else//ensure sum of probability = 1
        {
            rental_prob = 1;
            for(int temp = 0; temp < start; temp++)
                rental_prob -= Poisson(temp, rent_lam);
        }

        //returning probability
        if(end != 20)
        {
            return_prob = Poisson(returned, rent_lam);
        }
        else//ensure sum of probability = 1
        {
            return_prob = 1;
            for(int temp = 1; temp < returned; temp++)
                return_prob -= Poisson(temp, rent_lam);
        }
        probs[rented] = return_prob*rental_prob;
    }
    else
    {
        probs[rented] = 0;
    }
}
```

Figure 4. C program to calculate the rental-returning probability at one location

An interesting thing is that the reward R depends only on X_1^{rented} and X_2^{rented} being conditional on X^{move}, therefore, the state-transition probabilities can be derived by the following calculation

$$Pr\{S_{t+1}|S_t = s, A_t = a\} = \sum_{x_1^{rented}} \sum_{x_2^{rented}} Pr\{X_1^{rented}, X_2^{rented}, X_1^{return}, X_2^{return}|S_1, S_2, X^{move}\}$$

And Expected rewards for the state-action-next-state are

$$\begin{aligned} r(s, a, s') &= \mathbb{E}[R_{t+1}|S_t, A_t = a, S_{t+1} = s'] \\ &= \sum_{x_1^{rented}} \sum_{x_2^{rented}} r \times Pr\{X_1^{rented}, X_2^{rented}, X_1^{return}, X_2^{return}|S_1, S_2, X^{move}\} \end{aligned}$$

After calculating $Pr\{S_{t+1}|S_t = s, A_t = a\}$ and $r(s, a, s')$, we are ready to do "Value iteration"

$$\begin{aligned} V(s) &= \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')] = \sum_{s'} \left(\sum_{r} r\, p(s', r|s, a) + \gamma V(s') \sum_{r} p(s', r|s, a) \right) \\ &= \sum_{s'} r(s, a, s') + \gamma V(s') p(s'|s, a) \end{aligned}$$

To update the optimal policy, the “Value iteration” is considered with a little modification. Instead of computing the policies at the end of the algorithm, they are updated through iterations accompanying with the state-values. This computation inconsiderably affects the computational time because the state-values have been predetermined. In addition, the state-values are estimated at all actions, that means it does not depend on the policy of previous iterations, Figure 6.

```
float* probs1 = prob_Slot(startS1, endS1, 3, 3);
float* probs2 = prob_Slot(startS2, endS2, 4, 2);
float prob = 0;
float expected_reward = 0;
for(int rented1 = 0; rented1<=startS1; rented1++)
{
    for(int rented2 = 0; rented2<=startS2; rented2++)
    {
        prob += probs1[rented1]*probs2[rented2];
        expected_reward += (rented1 + rented2)*10*probs1[rented1]*probs2[rented2];
    }
}
```

Figure 5. C program to calculate transition probability and expected rewards for the state-action-next-state

```
float delta = 0;
for(int i = 0; i < n; i++)
{
    for(int j = 0; j < n; j++)
    {
        //estimate value
        float Sa[11];
        for(int a = 0; a < 11; a++)
        {
            int move = action[a];
            Sa[a] = -2*abs(move)*policy(i, j, move)*policy(j, i, -move);
            for(int h = 0; h < n; h++)
            {
                for(int k = 0; k < n; k++)
                {
                    float* prob_reward = transition_prob(S[i][j], S[h][k], move);
                    Sa[a] += (prob_reward[1] + prob_reward[0]*gamma*S[h][k]->v);
                }
            }
        }
        //improve policy
        float V = S[i][j]->v;
        float max = Sa[0];
        for(int a = 0; a< 11; a++)
        {
            if(Sa[a] >= max)
            {
                S[i][j]->v = Sa[a];
                S[i][j]->action_value = action[a];
                max = Sa[a];
            }
        }
        //terminate
        float error = abs(V - S[i][j]->v);
        if(delta < error) delta = error;
    }
```

Figure 6. C program to perform “Value iteration” Algorithm

3. Experiments and Results

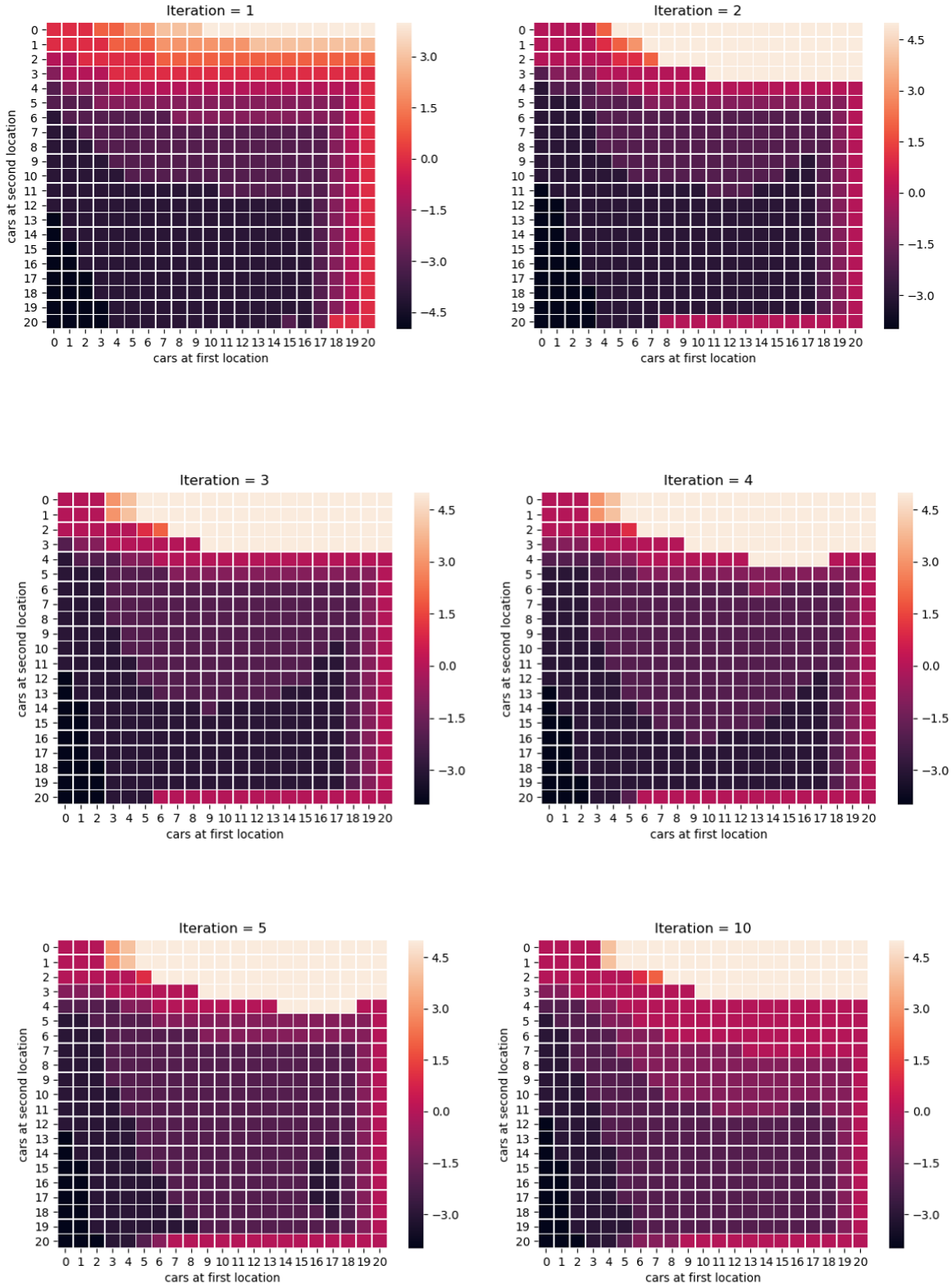

Figure 8. Heatmap of taken action in each state. At each state, the policy helps to decide how many cars should be moved and which location is the destination.

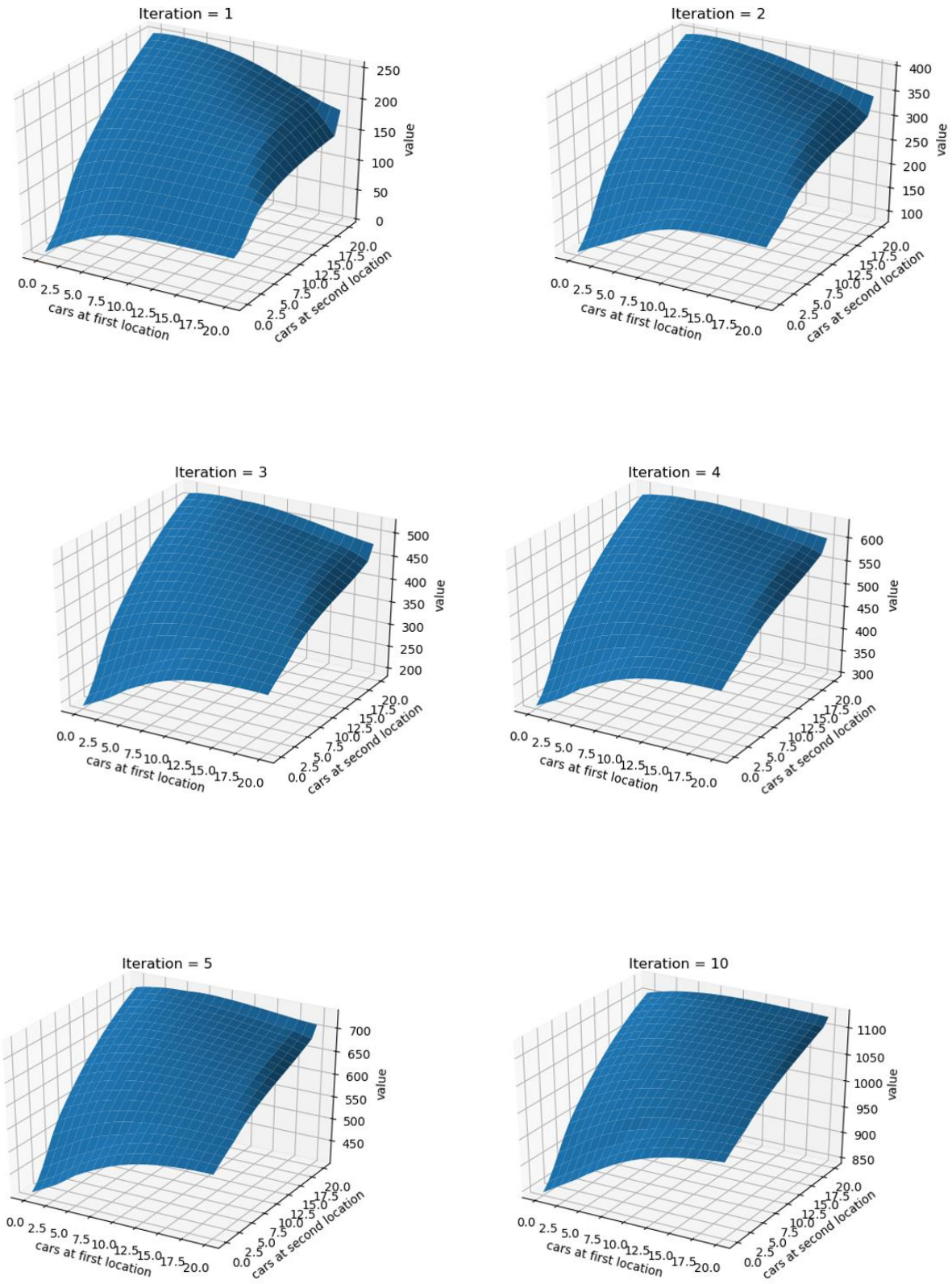

Figure 9. The estimated state-value at each state from 1 to 5, and 10 iterations.